CHKPLUG: Checking GDPR Compliance of WordPress Plugins via Cross-language Code Property Graph

Faysal Hossain Shezan
University of Virginia
fs5ve@virginia.edu

Zihao Su
University of Virginia
zs3pv@virginia.edu

Mingqing Kang
Johns Hopkins University
mkang31@jhu.edu

Nicholas Phair
University of Virginia
np4ay@virginia.edu

Patrick William Thomas
University of Virginia
pwt5ca@virginia.edu

Michelangelo van Dam
in2it
michelangelo@in2it.be

Yinzhi Cao
Johns Hopkins University
yinzhi.cao@jhu.edu

Yuan Tian
University of California, Los Angeles
yuant@ucla.edu

***Abstract*—WordPress, a well-known content management system (CMS), provides so-called plugins to augment default functionalities. One challenging problem of deploying WordPress plugins is that they may collect and process user data, such as Personal Identifiable Information (PII), which is usually regulated by laws such as General Data Protection Regulation (GDPR). To the best of our knowledge, no prior works have studied GDPR compliance in WordPress plugins, which often involve multiple program languages, such as PHP, JavaScript, HTML, and SQL.**

In this paper, we design CHKPLUG, the first automated GDPR checker of WordPress plugins for their compliance with GDPR articles related to PII. The key to CHKPLUG is to match WordPress plugin behavior with GDPR articles using graph queries to a novel cross-language code property graph (CCPG). Specifically, t he C CPG m odels b oth i nline l anguage integration (such as PHP and HTML) and key-value-related connection (such as HTML and JavaScript). CHKPLUG reports a GDPR violation if certain patterns are found in the CCPG.

We evaluated CHKPLUG with human-annotated WordPress plugins. Our evaluation shows that CHKPLUG achieves good performance with 98.8% TNR (True Negative Rate) and 89.3% TPR (True Positive Rate) in checking whether a certain WordPress plugin complies with GDPR. To investigate the current surface of the marketplace, we perform a measurement analysis which shows that 368 plugins violate data deletion regulations, meaning plugins do not provide any functionalities to erase user information from the website.

I. INTRODUCTION

WordPress, a well-known content management system (CMS), provides so-called plugins and themes—usually developed by third parties —for website owners to augment the default functionalities of the CMS. To date, WordPress has around 60K plugins [39] and they generate over a billion dollars of revenue each year [12]. One challenging problem of using WordPress plugins is that they may collect and process personally identifiable information (PII), which is regulated by privacy laws such as the European Union-introduced General Data Protection Regulation (GDPR). These plugins are published in the global App market and can be accessed or integrated by anyone around the world using the WordPress store. That is unless they specifically block EU traffic, it is the default assumption that they will have potential European Union traffic [32] and need to obey GDPR. An existing article [32] shows that it is the site owner's responsibility to ensure all the plugins installed on their website follow GDPR. Compliance with GDPR will help plugin developers to increase possible installations of their plugins. Plugin developers often are not familiar with privacy laws and therefore the need for an automatic checker is increasing for both plugin developers and website owners using these plugins.

To the best of our knowledge, there are no prior works that check the compliance of WordPress plugins against different GDPR articles related to private data. On one hand, researchers are studying the compliance of websites against certain GDPR articles, such as cookie and tracking opt-outs [73], [74]. Similarly, a multitude of free and paid services exist to evaluate the GDPR compliance of given websites and vary in complexity, ranging from consulting to cookie analyzers [5], [8], [16] to do-it-yourself checklists [7], [19]. However, these only cover a specific subset of GDPR requirements such as cookie consent or involve slow or expensive manual review [6], [17].

On the other hand, prior works investigated GDPR compliance in mobile app markets by checking the presence of user consent [67], [68], [83], privacy policies [26], [27], [56], [81], [89], [90], cookies [30], [37], and by analyzing network traffic data [41], [46]. While such works are important and successful in checking client-side GDPR compliance, unfortunately, they cannot be extended to WordPress plugins where PIIs are often collected on the client side using HTML and JavaScript, but processed at the server via PHP and SQL. That is, personally identifiable information is flowing between the client and server and is processed heterogeneously across the program language boundaries.

Such cross-language dataflows are challenging to identify, let alone be used for detecting GDPR violations because of

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24610
www.ndss-symposium.org

the variety and involvement of different program languages. For example, JavaScript code is interacting with HTML using Document Object Model (DOM) APIs, but with the server-side PHP using asynchronous HTTP requests and responses. At the same time, PHP is interacting with HTML and JavaScript using statements, such as "echo", but with JavaScript through HTTP requests and responses as well. Such heterogeneous dataflows make it challenging to extract PII and track its flow across languages.

In this paper, we design and implement CHKPLUG, the first automated GDPR checker of WordPress plugins. CHKPLUG identifies compliance with GDPR articles related to PIIs and processing of PIIs (*i.e.,* data access, data deletion, data sharing, and Security of PII) [1]. We select these GDPR articles based on the following criterion: (1) common articles that are shared by other privacy laws, e.g., California Consumer Privacy Act (CCPA) [2], and (2) articles that are directly relevant to data processing via computer programs. The key insight of CHKPLUG is to match WordPress plugin behaviors—represented as data- and control-flows and extracted via cross-language static analysis—against a set of predefined rules of GDPR articles. CHKPLUG reports a violation if a match is not found for a GDPR article. As an example, consider data deletion. CHKPLUG checks the lack of data flow between PIIs and deletion APIs as a violation of the data deletion article. Similarly, consider data sharing. CHKPLUG looks for privacy policy when PIIs are shared with a third-party via network communication such as HTTP requests.

CHKPLUG tackles and advances the cross-language challenge by representing WordPress plugins involving four different languages (JavaScript, HTML, PHP, and SQL) as a cross-language code property graph structure. That is, CHKPLUG first generates intra-language graphs (e.g., CPGs and DOM trees) and then connects different graphs across language boundaries into a CCPG. For example, CHKPLUG connects HTML DOM nodes with JavaScript variables using key-value analysis, i.e., if the HTML node ID matches the JavaScript DOM function parameter. For another example, CHKPLUG performs an inline analysis to generate placeholders for PHP code embedded as part of HTML and creates dataflows. Once CHKPLUG generates CCPG, it queries the CCPG starting from PIIs and tracks the PIIs flows for GDPR violations.

Ground-truth evaluation on 200 plugins shows that our tool achieves a 98.8% TNR (true negative rate) and a 89.3% TPR (true positive rate) for detecting violations of P_{access}, P_{delete}, P_{share} and $P_{security}$. To understand the current compliance situation of plugins, we perform a measurement analysis and run CHKPLUG on 2,722 plugins from the marketplace (Section IV). CHKPLUG determined 381 plugins (14%) to be non-compliant with GDPR regulations (Section V-C). Among them 368 plugins violate P_{delete}, 19 plugins violate P_{share}, and 36 plugins violate $P_{security}$. We find no violations for P_{access} because most of the time the plugins store PII data using WordPress core database, and the rest of the time they provide export functionality.

Contribution. We make the following contributions in designing and implementing CHKPLUG.

- **Cross-language analysis.** We perform cross-language analysis, including HTML, JavaScript, PHP, and SQL platforms. This helps us to capture all of the dataflows across different platforms.
- **Measurement analysis.** We perform a measurement analysis on the current WordPress plugin marketplace to identify the total number of plugins violating GDPR.
- **New tool.** We are the first to propose a generic framework called CHKPLUG, which automatically detects GDPR violations on websites. We implement a prototype of CHKPLUG that checks compliance with GDPR laws in WordPress plugins. We share our tool at https://github.com/faysalhossain2007/CHKPLUG.
- **Dataset.** We manually label 200 plugins. We publicly release our labeled dataset in https://github.com/faysalhossain2007/CHKPLUG.

II. A MOTIVATING EXAMPLE

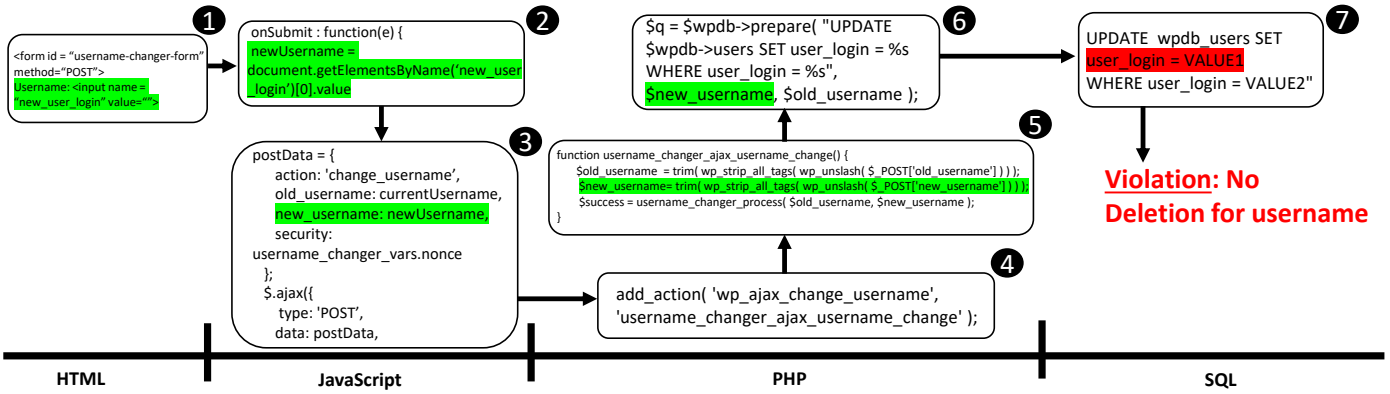
In this section we provide a motivating example which is found by CHKPLUG to be in violation of GDPR article 17, i.e., right to erase (a.k.a. right to be forgotten). The violation is located in a popular WordPress plugin, called Username Changer[2], which has an active installation of 30,000+ users and an average rating of 4.5 stars. The plugin provides an easy way to change usernames via a nice interface. Note that we have responsibly reported the GDPR violation to the plugin's developer, but have not received any feedback yet at the time of our paper submission.

Figure 1 (a) shows how this WordPress plugin collects and stores private data, but does not provide any deletion methods. The collection has four steps and spans four different programming languages. First, the plugin collects private user inputs (e.g., username, display name, and nickname) via an HTML form with an id "username-changer-form". The collected data is considered as Personal Identifiable Information (PII) according to the literature [1]. Second, the JavaScript code of the plugin reads the data using DOM operations and sends the data via a 'POST' request to the server-side plugin. Third, the server-side code, written in PHP, obtains the data and prepares SQL statements. Lastly, the SQL statements store the private data in the database. This collection and storage is a violation of GDPR right to erase because the plugin does not provide a means to delete the data. Instead, the data is stored in the server database after being collected.

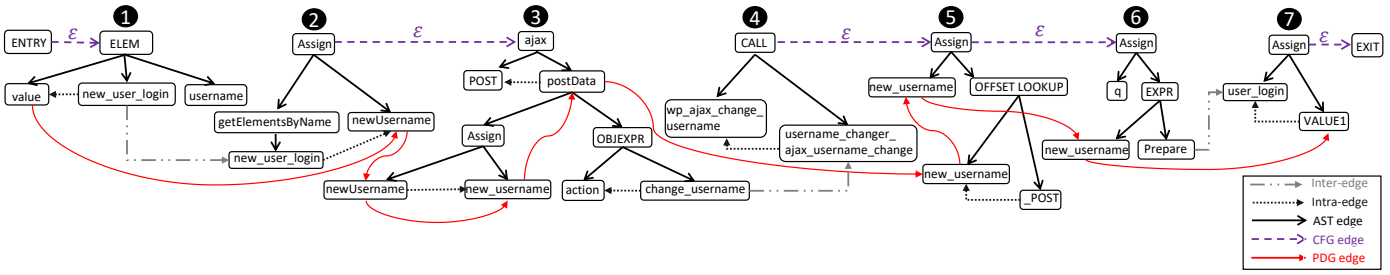
The major challenges in detecting this GDPR violation are the detection of PII collection and the lack of deletion of stored PII: The combination of these two conditions leads to such a violation. The PII collection detection is challenging because the private data flows across different language boundaries. That is, the dataflows between languages are heterogeneous: For example, HTML→JavaScript is based on a DOM operation, and JavaScript→PHP is based on HTTP requests. Furthermore, there are others that are not shown in the example, such as PHP→JavaScript via the echo statement. Similarly, the detection of lack of deletion is challenging because a deletion may exist across different languages and be controlled by a user on the client side.

[1]In this paper, we refer to data access as P_{access}, data deletion as P_{delete}, data sharing as P_{share}, and security of PII as $P_{security}$

[2]https://wordpress.org/plugins/username-changer/



(a) An illustration of the GDPR violation using code from different languages.



(b) Cross-language Code Property Graph (CCPG) Representation of the source code presented in Figure 1a.

Fig. 1: Motivating example for detecting data deletion violation in WordPress plugin.

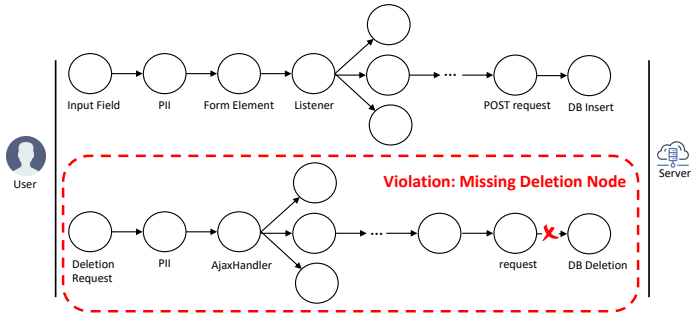


Fig. 2: Queries used for identifying data deletion violation.

Solution Overview Succinctly, CHKPLUG models WordPress plugins as a cross-language code property graph and violations of GDPR clauses as graph queries. Then, CHKPLUG reports a GDPR violation if a match is found in the graph.

Figure 1 (b) shows the CCPG of our motivating example. There are three different types of inter-language dataflows that are represented in the graph. First, the data flow crosses from the HTML form node (i.e., the value of the "input" node with a name as "new_user_login") to the return value of a JavaScript function call, `getElementsByName`. Second, the data flow crosses from JavaScript to PHP. Specifically, when JavaScript calls `$.ajax` with a key-value pair `new_username:newUsername`, the PHP code receives the data via `$POST`. CHKPLUG connects the JavaScript CPG node with the corresponding PHP CPG node based on the key "new_username". Lastly, the data crosses from PHP to SQL. CHKPLUG connects two nodes, one in the PHP CPG and the other in the parsed SQL statement, via the `prepare` statement in PHP.

Once CHKPLUG creates a CCPG such as the one shown in Figure 1 (b), the next step is to query it for GDPR violations. Such a query might follow a pattern like the one shown in Figure 2. Precisely, the query has two parts. The top shows that CHKPLUG finds a PII coming from an HTML input field and being stored at the server-side database. Then, the bottom shows that CHKPLUG finds that the deletion request is not connected with the database deletion, i.e., the PII is kept in the server-side database, leading to a violation of GDPR's right to erase in its article 17. Note that the top is a prerequisite of the bottom: If PII is not stored, CHKPLUG will not look for deletion and possible GDPR violations.

III. SYSTEM DESIGN

We introduce the design of CHKPLUG in this section.

A. Overall System Architecture

Figure 3 shows the overall system architecture of CHKPLUG, which takes the source code of WordPress plugin as input and outputs a list of GDPR violations. CHKPLUG has three major components: (1) parser, (2) CCPG generator including (2.1) intra-language edge connector and (2.2) cross-language integrator, and (3) GDPR compliance checker.

Here are the detailed descriptions of the three components. First, the parser is to detect and parse language-specific source code. Since the parser is standard and we do not claim any contributions, we leave the descriptions to Section IV-A. Second, CHKPLUG generates CCPG via two steps: (i) intra-language CPG generation (especially event-related control- and data-flows), and (ii) cross-language CCPG generation.

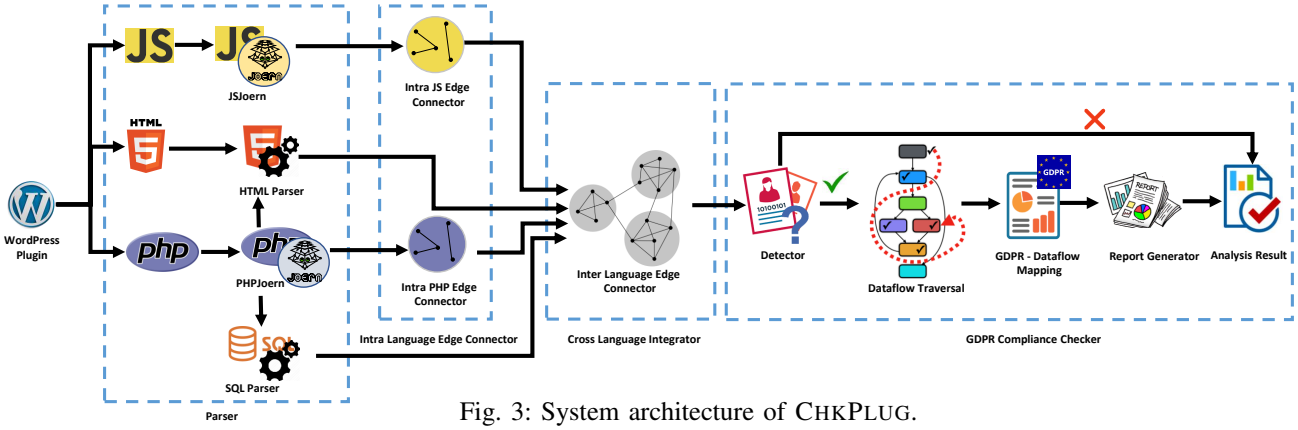


Fig. 3: System architecture of CHKPLUG.

Lastly, CHKPLUG traverses all paths involving PII operations to identify the requirements according to GDPR policy. For example, if a plugin stores data in the database, it must provide data deletion functionality. Here, we map all of the requirements for GDPR policies along with the generated graph using our GDPR compliance checker module (Section III-C). We conclude with the analysis result listing whether a particular plugin violates GDPR.

B. CCPG Generation

In this subsection, we describe how CHKPLUG generates CCPG via both inter- (i.e., cross-) and intra-language analysis.

1) Cross-language Analysis: CHKPLUG's cross-language analysis depends on different language pairs and adds customized intra-language dataflow edges. From a high level, cross-language analysis can be classified into two major categories: (i) inline analysis and (ii) key-value analysis. Let us explain these two. First, inline analysis is based on the fact that the statements of one language are embedded into another language. For example, PHP can be embedded as part of an HTML file and outputs HTML via the "echo" statement. Similarly, PHP can prepare SQL statements and then execute prepared SQL statements with customized values. CHKPLUG performs inline analysis by first analyzing the embedded statements (e.g., PHP in the PHP-HTML case). Then, CHKPLUG replaces the outputs of the embedded language statements with pseudo placeholders and then analyzes the latter language (e.g., HTML in PHP-HTML case) to generate intra-language dataflows.

Second, key-value analysis is for these program languages that are connected via APIs (e.g., HTML and JavaScript) or network protocols (e.g., JavaScript and PHP). For example, when JavaScript sends an HTTP POST/GET request to the server, the server-side PHP code receives the values in either `$POST` or `$GET` variables. CHKPLUG performs key-value analysis via first matching keys at both languages and then connecting the corresponding values with inter-language dataflows. For example, CHKPLUG looks for the keys in `$ajax` calls at client-side JavaScript and matches the corresponding keys in `$POST` at PHP, and then creates dataflows between both values.

Now, we illustrate several language pairs and describe how CHKPLUG performs the aforementioned inline and key-value analysis to create cross-language dataflows using a few examples under each pair.

```
1 <p> a <?php echo 'dog' ?> </p>
```

Listing 1: PHP code embedded inside HTML.

HTML⟷PHP On one hand, PHP can be embedded as part of HTML and output HTML code; on the other hand, HTML can be used to send HTTP requests to PHP. CHKPLUG adopts inline analysis for the former and key-value analysis for the latter. We describe the details below:

- Inline PHP Traversal (PHP→HTML). CHKPLUG connects dataflow edges from the AST of the inline PHP to the corresponding placeholder that represents the PHP output. For example, in Listing 1, CHKPLUG connects the root node that corresponds to *echo 'dog'* to a corresponding text node in HTML that represents *<? php echo 'dog' ?>*.
- HTML Form Submission (HTML→PHP). A common type of communication between HTML and PHP is via form submission, in which the input data in a form is submitted and is handled by a `POST`, `GET`, or `REQUEST` variable in PHP. CHKPLUG connects dataflow edges between every input field (e.g., <input name = "foo" .../>) in HTML and places the input field that is referenced in PHP (e.g., $_POST['foo']).

PHP⟷SQL Data can be passed from PHP code to a SQL query execution that stores data into a database (e.g., through an INSERT statement); then, similar to SQL data storage, data can also be retrieved from databases via SQL queries, and then the data is received by PHP code (e.g., a variable). CHKPLUG performs these two types of inline analysis.

- SQL Data Storage (PHP→SQL). CHKPLUG connects dataflows from data supplied to SQL query function call in PHP to the SQL AST for data storage SQL queries (e.g., connect a dataflow edge from data that is passed to the INSERT statement to the SQL AST for the INSERT statement).
- SQL Data Retrieval (SQL→PHP). CHKPLUG connects dataflows from the SQL AST to the PHP endpoint that receives the query result for data retrieval queries (e.g., connect a dataflow edge from the SQL AST of a SELECT

```
1 <?php $pass='123';?>
2 <script>var jspass='<?=pass?>';</script>
```

Listing 2: Connecting PHP and JavaScript.

```
1 $foo = esc_html('<p>hello</p>');
2 $bar = 'a' . 'b' . $foo;
```

Listing 3: Resolving value using value resolver.

statement to the PHP variable that receives the queried result).

PHP⟷JavaScript PHP can be interpreted to yield JavaScript just as it can be interpreted to yield valid HTML. This allows data contained in PHP code to propagate to JavaScript expressions. At the same time, WordPress provides an Ajax Handler that receives Ajax requests from JavaScript, processes `$POST` or `$GET`, and then fires a WordPress Action along with the data sent from the Ajax requests. Then, plugins can receive the data in PHP via hooking a PHP handler to the fired Action.

- Inline JavaScript Traversal (PHP→JavaScript). CHKPLUG connects edges between the PHP nodes and a placeholder that represents the resultant JavaScript. For instance, given the code snippet listed in Listing 2, we connect the PHP node for the variable to the node for its usage in JavaScript.
- WordPress Ajax Handler (JavaScript→PHP). CHKPLUG models this data transmission logic by first locating Ajax requests that send data to the WordPress Ajax handler endpoint, and then analyzing the actions that are going to be fired by WordPress. Next, CHKPLUG analyzes the PHP functions that are hooked as callback functions to the fired actions. With these, CHKPLUG matches the data being sent out by the Ajax requests and the data being received in the PHP functions and connects data flow edges between them.

HTML⟷JavaScript JavaScript and HTML are communicating with each other via different types of DOM APIs. The connections are done via two types: DOM events and HTML selectors (such as IDs, names, and class names). Once JavaScript obtains a DOM element, it can either read its data or write to it via properties like `innerHTML`. We now describe these two types.

First, DOM events may trigger JavaScript callbacks and interact with HTML. Let us use submit events as an example. When the onsubmit event of HTML elements is registered in JavaScript, CHKPLUG creates dataflow edges between the form node in the HTML and the root node of the handler code.

Second, JavaScript may use selectors, e.g., jQuery or built-in selectors, to obtain HTML elements. Take jQuery selectors for example. CHKPLUG parses the selector statements, maps the selector to the HTML AST, locates the selected HTML elements, and next connects data flow from the located HTML elements to the selectors with keys such as ID, names, and class names. After query selecting, CHKPLUG also finds all usages of dataset objects and connects the usage to the actual data attributes in the HTML graph.

2) Intra-language Analysis: Our intra-language analysis follows traditional static analysis in generating different control- and data-flow edges. Prior research has shown this and we do not claim any contributions [24], [28]. Below, we introduce two types of control- and data-flow edges that are specific to WordPress or are traditionally challenging to resolve.

- *Hook-related call edges.* WordPress provides an event listening system called *hooks*, where plugins can either trigger events and send out data (through do_action() or apply_filter()) or register callback functions that listen to events and receive data sent out by the event trigger (through add_action() or add_filter()). CHKPLUG models hooks by connecting dataflows from the data sent out by the event trigger to parameters of callback functions that are registered to receive such data. In this way, CHKPLUG determines the triggered functions, and thus traces the personal data flow.
- *Control- and data-flow edges related to object properties.* Method calls like $obj->$foo()) and property accesses like $obj->$var or $this->$var are challenging to resolve, because the host object's types are usually unknown due to the dynamic nature of PHP. CHKPLUG traces back the constructions of such objects to find their types and checks the official doc comments (e.g., PHPDoc) for functions or class variable declarations.
  There are two steps. First, CHKPLUG identifies the parent-child relationship across all classes in the PHP source code. That is after the CPG is constructed, CHKPLUG identifies all function calls using $self and $parent variables (*e.g.,* $self::foo()) and connects call edges based on overriding principles. In addition, CHKPLUG connects dataflow edges from the definitions of class constants or static variables (*e.g.,* connect edge from const CONSTANT = 1 to $self::CONSTANT.
  Second, once the class hierarchy is constructed, CHKPLUG determines object types and builds edges for method calls and class variables in an iterative process, until results converge and no new edge can be constructed.
- *Value resolver.* We build our resolver component to determine the values of variables that store URLs, HTML scripts, or SQL queries inside PHP. It statically determines the values for variables in case they are assigned values of basic data types (*e.g.,* string, integer, float). Specifically, for a given variable node, CHKPLUG first backtracks dataflow edges to all places they are being assigned. After tracing back to a variable's assignment, CHKPLUG traverses the AST to identify the data that is being assigned to the variable, and their relationships with each other (*e.g.,* they are concatenated together). So that CHKPLUG can resolve the complete value for that variable. Furthermore, CHKPLUG captures the behaviors of string filtering and sanitization functions provided by WordPress (*e.g.,* esc_html(), $wpdb->$prepare()) to accurately resolve string values. However, if part of a variable's value does not have any resolvable value (*e.g.,* the value comes from a database), CHKPLUG uses a dummy value instead to complete the resolving process.
  To better illustrate, let us consider the example in Listing 3. If the program tries to resolve the value for $bar, it will trace back to its assignment, and find that the

value is the result of a concatenation among 'a', 'b', and ' hello ' (the value of $\$foo$ after filtered through esc_html()). Therefore, the resolved value is 'ab hello> '.

- *Object type determination.* CHKPLUG statically determines the class/type of an object. CHKPLUG first produces a representation of all classes and their method calls and class variables. Then it parses PHPDoc comments and strict typing for method declarations and variable declarations. While determining the type of an object, CHKPLUG first backtracks to a node with types already recorded, and then checks the recorded type. However, if this process fails, CHKPLUG also attempts to backtrack to where the object is being created (*e.g.,* $\$obj$ = new Foo();) or initialized to determine its type. In this way, it obtains the types for class variables, arguments, and return values of methods.

C. GDPR Compliance Checker

Once CHKPLUG finishes constructing the CCPG, it starts to gather information to identify GDPR compliance. Specifically, we divide the GDPR compliance checker into three connected sub-components: (i) detectors, (ii) graph query for GDPR violation, and (iii) violation report generator.

1) Detector: The purpose of the detector is to find three types of nodes: source, sink, and security. *Source nodes* are those that include PII collection/retrieval functionalities. *SECURE* nodes are those intermediate nodes that run security/hashing mechanisms on the inputted data. *Sink node* are those related to the processing of PIIs, e.g., those points where personal data is being stored, updated, deleted from the database, or sent to a remote address. Let us describe the detection of these three types of nodes below.

Source Detector CHKPLUG identifies three different types of source nodes from where personal data can be collected or retrieved i.e., (1) HTML form inputs, (2) WordPress core database, and (3) custom database.

- HTML Form Inputs. The most common way of collecting user information is through a user interface written in HTML. With such an interface, a plugin can collect various personal information, including but not limited to a user's email, name, password, location, address, and date of birth. CHKPLUG finds those entry points by querying HTML form input nodes and identifying ones with displayed field names that contain personal data keywords using a manually crafted rule-based approach (Appendix B).
- WordPress core database. Plugins can retrieve personal data from WordPress's core database tables which contain user data and are shared across plugins. For this, CHKPLUG considers two cases: (1) the retrieval function directly gets personal data (*e.g.,* get_userdata($user_id) gets the user's profile data such as email and name), or (2) the retrieval function gets user data based on a key that has a matching personal data keyword (*e.g.,* get_user_meta($user_id, 'shipping_address') gets the user's shipping address, which is previously stored as the user's metadata). CHKPLUG keeps a record of the argument position of the key for each function based on the WordPress documentation locally in a file for mapping purposes.
- Custom database. Plugins can retrieve personal data from database queries (*e.g.,* SELECT query) on tables that contain personal data. This database is specific to each plugin and is not shared across other plugins. CHKPLUG first scans for all database table creation queries and labels tables with fields that have matching personal data keywords (Appendix B). Note that CHKPLUG will not be able to detect storage where the table's field name does not contain any matching personal data keyword but is used for storing personal data. Next, CHKPLUG scans for all database queries that retrieve data from the tables that record personal data.

Security Detector GDPR laws require encrypting personal data before sending it over the network (either to first- or third-parties). To identify the existence of such functionalities in a plugin's code, CHKPLUG analyzes the presence of encryption or hashing functions. We label nodes that perform security operations (encryption or hashing) on personal data as SECURE. We discuss the list of functions that CHKPLUG recognizes in Appendix D.

Sink Detector CHKPLUG finds four types of sink nodes that perform operations related to (1) WordPress core database, (2) custom database, (3) remote request, and (4) file storage. We discuss those in detail below.

- WordPress core database. Plugins can store, retrieve or delete data from WordPress's core database tables (*e.g.,* add_user_meta() stores metadata for a user. Refer to Appendix G for the list of WordPress data storage and deletion functions). We identify and mark all WordPress-provided function calls as sinks because they can potentially process personal data depending on the inputs. In addition, we label the argument inputs of these function calls that are either the data or the key used in the database operation.
- Custom database. Plugins can store or delete data through queries (*e.g.,* INSERT and DELETE statements) on tables that contain personal data. We identify all such database operations and record their operation type, table name, and fields. Furthermore, we backtrack the dataflow edge from SQL to PHP (as explained in Section III-B1) to identify the PHP node that passes data to the SQL query and executes the query, and then we mark it as a sink node. For example, in Figure 1, *username* information is passed to the SQL query to update the user information.
- Remote request. Plugins send personal data to third-parties via remote requests (*e.g.,* curl). There are three ways to send such data in WordPress- (1) PHP native functions, (2) WordPress functions in PHP and (3) jQuery requests (*e.g.,* jQuery.post()) in JavaScript. We list the functions used for remote requests in Appendix E. For each such sink point, we identify the argument input that corresponds to the URL. We then perform backward dataflow traversal to resolve possible URLs (Section III-B2).
- File storage. In addition to storing data in the WordPress core database or custom databases, plugins can also store data in files. For this, it can use several PHP functions

(*e.g.,* using fput()). CHKPLUG scans for the usage of PHP native functions (Appendix F) for detecting nodes that store any data to files.

2) Graph Queries for GDPR Violations: The second subcomponent of the GDPR compliance checker is to conduct graph queries for GDPR violations based on different GDPR articles. Specifically, Table I listed the required dataflows that should be present in WordPress to comply with the corresponding GDPR policy. We now describe the details based on different policies separately.

[Article 15] Data Access (P_{access}). Article 15 mandates user access to stored personal data. If the plugin utilizes any custom database for storing personal data, the plugin is required to provide data export functionality to comply with P_{access}. Note that a plugin is not strictly required to provide data access functions, as WordPress natively provides an exporter tool that can access data from WordPress's core databases. Yet, WordPress still encourages plugins to implement a data access function as a best practice [23]. In summary, we devise the following rules integrated into CHKPLUG to determine whether a plugin follows P_{access}:

- A plugin stores PII via a custom database and the plugin provide the option to download all the stored PII → `COMPLY`
- A plugin stores PII in WordPress core database → `COMPLY`
- A plugin does not store PII → `COMPLY`
- A plugin stores PII in a custom database but provides partial/no set of PII to export → `VIOLATION`

While data can be exported in many ways, CHKPLUG considers the sink nodes based on WordPress's official privacy guidelines [23]. WordPress provides website owners with a privacy tool to manage data export and gives plugins an interface to supply the exported data. CHKPLUG specifically checks if each type of personal data stored is supplied to the interface. To achieve this, for each type of storage method, CHKPLUG searches for a counterpart method that can retrieve the data from the storage method as listed in Table I. For a WordPress storage function, the counterpart data retrieval function has a matching data type and data key. For example, if a plugin stores email through update_user_meta($id, 'email', $email), the corresponding retrieval function would be get_user_meta($id,'email'), as it operates on the matching data type user_meta, and has the same data key as 'email'. For a database operation, the counterpart deletion function has a matching table name. For example, if a plugin stores personal data in table tab1 through an *INSERT INTO* statement, the corresponding retrieval function would be a *SELECT* statement operating on the same table with the same personal data. In Table I, we can observe that P_{access} involves two types of actions, store, and export. Store actions indicate the process of collecting PII either from the user interface or database. The action inserts that particular PII into the database. However, export actions refer to the presence of retrieval functionalities. In this process, CHKPLUG seeks the database retrieval for that PII.

More specifically, if a plugin complies with P_{access}, for each instance of data storage nodes, CHKPLUG searches for instances of counterpart retrieval node that has a data flow path to the exported data supplied to WordPress. This indicates that the personal data is retrieved and supplied to WordPress's exporter tool. If there is at least one storage node that has no counterpart retrieval node supplied to WordPress, CHKPLUG marks the plugin as violating the P_{access}.

[Article 17] Data Deletion (P_{delete}). P_{delete} is almost similar to P_{access}, except for P_{delete} plugin needs to provide deletion functionality even for the storage in WordPress core database according to article 17. We list the following rules to identify whether a plugin violates P_{delete}.

- A plugin stores PII and provides the option to delete all the stored PII → `COMPLY`
- A plugin does not store any PII → `COMPLY`
- A plugin stores PII but only provides partial or no set of PII to delete → `VIOLATION`

Similar to the analysis for P_{access}, for each type of storage method, we define a counterpart method that can delete the data from the storage method. Table I illustrates the required two flows, one for store and another for delete. These are necessary for a plugin to comply with P_{delete}. For a WordPress storage function, the counterpart deletion function has a matching data type and data key. For example, if a plugin stores email data through update_user_meta($id,'email', $email), the corresponding deletion function would be delete_user_meta($id,'email'), as it operates on the matching data type user_meta, and has the same data key email. For a database operation, the counterpart deletion function has a matching table name. For example, if a plugin stores personal data into table $tab1$ through an *INSERT INTO* statement, the corresponding deletion function would be a *DELETE* statement operating on table $tab1$.

To be specific, if a plugin complies with P_{delete}, for each instance of a data storage node, we search for instances of counterpart deletion node. If there is at least one storage node that has no counterpart deletion node, we determine the plugin as violating P_{delete}.

[Article 28] Third-party Data Sharing (P_{share}). The plugin is responsible for disclosing third-party data sharing to users. If CHKPLUG finds any remote request sink that personal data sources can traverse to, it implies that the plugin sends certain personal data to a third party. In such a case, the plugin is required to comply with P_{share} according to article 28. If plugins do not send PII to a third party, then they do not need to follow P_{share}, and thus are automatically compliant. Whenever the plugin is sharing data with a third party it needs to disclose it in its privacy policy. Failing to do so will result in a GDPR violation. Moreover, P_{share} applies regardless of the URL, because even if the receiving endpoint is the plugin developer, such a case is still considered third-party data sharing, as the plugin developer is considered a third-party from the perspective of the website owner that deploys the plugin in their website. In particular, we build the following rules to check violation of P_{share} using CHKPLUG:

- A plugin does not collect any PII → `COMPLY`
- A plugin does not share any PII → `COMPLY`
- A plugin shares PII with a third party (including the plugin developer website) and discloses it in the privacy policy → `COMPLY`

TABLE I: GDPR policies covered by CHKPLUG.

ID	Action	Flow
P_{access}	Store	$collect_{HTML}(PII)\|\|retrieve_{DB}(PII) \rightarrow intermediate_{node_1,node_2,....,node_N} \rightarrow storage_{DB}(PII)$
	Export	$counter_{retrieveDB}(PII) \rightarrow intermediate_{node_1,node_2,....,node_N} \rightarrow export_{interface}(PII)$
P_{delete}	Store	$collect_{HTML}(PII)\|\|retrieve_{DB}(PII) \rightarrow intermediate_{node_1,node_2,....,node_N} \rightarrow storage_{DB}(PII)$
	Delete	$counter_{retrieveDB}(PII) \rightarrow intermediate_{node_1,node_2,....,node_N} \rightarrow delete_{interface}(PII)$
P_{share}	Send	$collect_{HTML}(PII)\|\|retrieve_{DB}(PII) \rightarrow intermediate_{node_1,node_2,....,node_N} \rightarrow remote_{request}(PII)$
	Disclosure	$doesExist(privacy-policy)\&\&disclose(sent_{PII}, remote_{URL})$
$P_{security}$	Send	$collect_{HTML}(PII)\|\|retrieve_{DB}(PII) \rightarrow secure_{node} \rightarrow remote_{request}(PII)$

- A plugin shares PII with a third party (including the plugin developer website) but does not disclose it in the privacy policy → `VIOLATION`
- A plugin shares PII and does not have any privacy policy → `VIOLATION`

As a result, we search for all the remote request sink points that send out PII and aggregate all personal data types sent out via these sinks. Note that we mark all remote requests as sending data to third parties, because a plugin developer, itself, is considered a third party to the website that deploys the plugin. That is why sending data to the plugin developer is considered third-party data sharing. We search for the flow of send and disclosure (Table I) in the plugin CCPG. Unique to analyzing P_{share}, we consider privacy policy to complete the flow for disclosure action. We extract the privacy text the plugin provides to WordPress through *wp_add_privacy_policy_content()* (WordPress's recommended way to provide privacy policy texts [22]). We then leverage existing work on privacy policy, PolicyLint [26] to analyze the privacy text and identify the list of personal data collected by third parties. Next, we compare this list with CHKPLUG's output. If the plugin fails to disclose any data collected by a third party (or does not have a privacy policy at all), we report that plugin as in violation of P_{share}.

[Article 32] Security of PII ($P_{security}$). According to article 32, plugins need to encrypt or perform hash operations on personal data before sending it over the network. Even if they use a secure channel for such communication, then it is considered protected. Failure to do so will violate $P_{security}$. We determine a plugin violating $P_{security}$ by checking the following rules-

- A plugin sends the encrypted PII to secure/insecure channel → `COMPLY`
- A plugin sends PII to any remote URL a secure communication channel (*e.g.,* HTTPS) → `COMPLY`
- A plugin sends PII to any remote URL an insecure communication channel (*e.g.,* HTTP) → `VIOLATION`

Both P_{share} and $P_{security}$ requirements are related to whether the plugin sends personal data to a remote URL. While many security settings are out of plugin developers' control, we search for direct evidence that the plugin is transmitting personal data via an insecure method.

Specifically, for each remote request sink point that personal data can traverse to, we parse the possible endpoint URLs which are previously analyzed by our detector (Section III-C1) and check whether the URLs use HTTP rather than HTTPS protocol. Furthermore, for each such sink point, we check if the node has a *SECURE* label, which would imply that the data is hashed or encrypted. If a plugin uses an insecure method to transmit personal data and does not secure the data in any way, it violates $P_{security}$.

3) Report Generator: After all analyses are done, CHKPLUG compiles a human-readable report that includes the final compliance decision and, if applicable, all evidence that a plugin violates GDPR. Moreover, to help developers comply with GDPR, CHKPLUG produces a fix report that guides plugin developers step by step to fix their violations. We have uploaded an example report to [15]. In the following, we illustrate how CHKPLUG generates compliance reports.

- Violation evidence. CHKPLUG outputs all evidence that shows the plugin is required to comply with a GDPR law, whereas the plugin lacks the actions to comply. Take the example shown below, for a piece of evidence that shows the plugin is not compliant with P_{delete}, CHKPLUG outputs the data storage sink of personal data that indicates the plugin needs to comply with P_{delete}, whereas the plugin has no corresponding deletion method. CHKPLUG indicates the code location where the developer can find this sink, as well as the personal data type that is being stored. We have illustrated a case in which CHKPLUG detects PII storage in a specific file of the plugin's source code. As a result, CHKPLUG is indicating the applicable GDPR policies for such storage.

  > [Art.17, Right to erasure] WordPress storage of PII through update_user_meta ($current_user → ID, 'last_name', $asmember_register_name) does not have a corresponding deletion method. Storage method found in file `templates/single-asmember-memberships.php` at line 771. Storage method stores user data of type: last name.

- Fix report. To help plugin developers comply with GDPR, CHKPLUG refers to the plugin handbook provided by WordPress [22] and provides a breakdown of all specific steps needed to comply with GDPR for all laws the plugin violates, along with auto-generated template code and *TODOs* inside the code based on the specific law requirements of the plugin.

With that, CHKPLUG determines whether a plugin complies with GDPR by analyzing the cross-language code property graph.

IV. IMPLEMENTATION AND DATASET

In this section, we describe our open-source implementation and dataset.

A. Implementation

Our implementation is open-source with 24,251 lines of code (LoC) *excluding* any third-party libraries or open-source tools. The implementation is available at: https://github.com/faysalhossain2007/CHKPLUG. We adopt an open-source HTML parser [11] and make changes with 1,190 LoC in Python. We also adopt an open-source SQL parser [18] and make changes with 436 LoC written in Python. Our implementation of JavaScript and PHP CPG is based on navex [24] and we make the following modifications.

- A customized JavaScript parser. We use Esprima to build a customized parser that converts JavaScript code to be accepted by navex.
- Intra-procedural data-flow edges. For a given REACHES edge (*i.e.,* a traditional data flow edge that connects data flows between two lines of code) between two lines of code, CHKPLUG searches for the corresponding nodes that represent the same data unit in the AST of the two lines of code and connect them.
- Hierarchical edges. CHKPLUG traverse the hierarchy of ASTs for each line of code and connect data units inside the AST, capturing and connecting movements of data units within a line of code.
- Function call and return edges. First, CHKPLUG finds call edges between different nodes. They are constructed between a function call and the function definition, and it is difficult to track the individual arguments passed to the function. As a result, CHKPLUG follows call edges and connects dataflows between arguments within a function call and the parameters inside the function definition. Second, following the logic for function call edges, CHKPLUG tracks how data is processed after it is passed to a function and then returned. Therefore, CHKPLUG follows call edges to locate all the return statements inside the function definition and connects dataflows for the returned data.
- Traversal of PII and Security Nodes. After constructing all dataflow edges and identifying all the PII sources and all security nodes that hash or encrypt data, CHKPLUG performs dataflow traversal using 'apoc.path.subgraphNodes', a Neo4J procedure that traverses all dataflow edge types we designate and outputs all reachable nodes within the graph. CHKPLUG uses the procedure to traverse from all the source nodes and get all nodes reachable through the dataflow edges that it generated. Therefore, CHKPLUG obtains all nodes that can possibly be reached by any of the personal data source or security nodes. Our tool labels all nodes that can be reached by a personal data source as *PERSONAL*, and all nodes that can be reached by a security node as *SECURE*, and CHKPLUG records down the corresponding sources that can reach to the node. Furthermore, CHKPLUG combines the personal data type of the personal data sources (*e.g.,* an array node may be reachable by a node with email data and another node with password data, and we mark the array node as containing email and password data). Similarly, for nodes reachable by security nodes, our tool combines the encryption/hashing methods from the sources. Next, CHKPLUG identifies all sinks among the nodes that are reachable by personal data sources, as these sinks conduct operations on the personal data sources; among these sinks, it also identifies the ones that are reachable by security nodes, as these sinks process encrypted or hashed data.

TABLE II: Ground truth labeled data of 200 plugins for evaluation. ✓= violation ✗ = non-violation.

#Violation	**#P_{access}**	**#P_{delete}**	**#P_{share}**	**#$P_{security}$**	**#Plugin**
✗	200	131	190	190	124
✓	0	69	10	10	76

Next, we also describe our manual efforts in identifying these source/sink/security functions for our analysis. Specifically, we inspect the behavior of APIs in different languages (e.g., DOM, PHP, and WordPress) manually. Let us use the WordPress APIs as an example. The WordPress team maintains a number of packages required for the proper operations of WordPress. These packages live in the `wp-include` directory of WordPress and can be used by plugin developers. Similarly, `wp-admin` contains functions related to administrative actions. To begin, we gather all of the functions defined in the `wp-admin` and `wp-include` directories of WordPress using our Python crawler. In total, there are 2,885 functions [21]. We then categorize these functions by behavior as either sources, sinks, or neither. Again, each function sets, deletes, or retrieves different user information. So, we label both operations and the types of collected user information manually. Before labeling, we read the complete descriptions of the WordPress functions from the official website. In total, we find 30 store, 65 retrieval, and 28 delete functions. We include the complete list in Appendix G. CHKPLUG uses these labeled functions while building the detector.

B. Dataset Collection

We collect WordPress plugins from the official website [4] in May 2022. To that end, we build a Selenium-based Python crawler that collects all the links of published plugins (so far) from https://plugins.svn.wordpress.org/. We visit each plugin's webpage (hosted in [4]). If a particular plugin is still active, then the user will be able to download it. Using our crawler, we are able to get the source code of all the plugins successfully. For each plugin, we collect the active installation numbers, last updated information, average ratings, current version, and plugin's code. We select 2,722 plugins from the marketplace. Out of these 2,722 plugins, we manually analyzed 200 plugins as ground truth for evaluating the performance of our tool.

1) Manual Labeling: Important to our evaluation is a ground truth with which we can compare the results of our system analysis. To establish this ground truth we manually analyze the plugin behavior and source code. Three computer science students read the GDPR extensively and consulted with a domain expert—who has six years (2016-) of experience in working with GDPR and privacy regulations—to learn the labeling methods. Later, they manually label those plugins, and whenever they have a conflict, they resolve it by discussing it with the domain expert. Each manual analysis aims to establish compliance or non-compliance concerning the same properties that CHKPLUG does. Those properties are data access, data deletion, data sharing, and security of PII. To begin, an analyst will install the plugin into a local version of WordPress. From there, they interact with the site and

plugin operating as a regular user would. The analysts are alert to all of the instances where the plugin would prompt for information. For any case where PII is collected, they then seek out the appropriate access and deletion endpoints. If no such interfaces are found, the plugin is said to be non-compliant. If found, a further analysis still must be done to validate the implementation. This requires inspecting the database backing the site. More particularly, the analyst will locate the records in the database added by the plugin. Database management tools with user-friendly front ends (*e.g.,* Adminers) make this process simple. An analyst can then confirm the plugin's compliance by ensuring the database is updated appropriately when interfaces like those to remove PII are invoked.

Inspecting the local database will not help identify cases of third-party information sharing. For these cases, the approach is to inspect the source code. One area to pay close attention to is including expressions. Third-party packages need to be included in the source file so their presence is a possible indication that user information may be shared externally. To confirm, it is helpful to load the plugin source into an *IDE* and navigate the call graph. Most IDEs have features to find usages of particular methods which can help trace data flow. Our analysis includes inspecting the source code in this way to identify violations. Lastly, we complement this static analysis with a more dynamic approach like using a debugger. If there are functions whose behavior cannot be determined or ambiguity in the code, we launch Xdebug, a popular PHP debugger. The debugger is connected to the WordPress instance and breakpoints are set in the areas of confusion. We could then interact with the plugin through the web interface, trigger the breakpoints, and gain access to data on the stack. This approach is helpful to confirm the exact information being transferred by the API calls. Ultimately, these methods are used to establish compliance concerning information sharing and information security.

While thorough, this process is very time-consuming. Often an analysis of a single plugin could take between 4-6 hours (approx.). In total, we manually inspect 200 plugins using this approach. Of these 200, 76 plugins violate one or many GDPR violations while 124 plugins comply with GDPR policies. Table II provides a breakdown of compliance across different GDPR policies. Note that we have not considered WordPress themes because those are for polishing the User Interface and website visual effects, which do not collect user data in general [20], [44]. We also manually verified 30 randomly-selected WordPress themes and found that none collect PII.

V. Evaluation

In the following section, we run a series of experiments to evaluate the performance of ChkPlug. In the end, we evaluate the end-to-end performance of ChkPlug (Section V-D). We run the experiments on AWS with 6 EC2 instances. These machines are of the t3.medium variety. They feature Intel(R) Xeon(R) Platinum 8259CL Processors, 2 vCPUs, 4 GBs of RAM, and run Amazon Linux 2.

A. Evaluation Questions and Metrics

To evaluate the performance of ChkPlug, we seek answers to the following questions:

TABLE III: RQ1: Detailed performance of ChkPlug on 200 plugins for identifying different GDPR violations. Here, TP= True Positive, FP = False Positive, TN = True Negative, and FN = False Negative.

Policy	TP	TN	FP	FN	TPR	TNR
P_{access}	0	200	0	0	100%	100%
P_{delete}	66	127	5	2	97%	96.2%
P_{share}	7	189	1	3	70%	99.5%
$P_{security}$	9	189	1	1	90%	99.5%
Total	**82**	**705**	**7**	**6**	**89.3%**	**98.8%**

- **RQ1.** What is the performance of ChkPlug in detecting GDPR violations in plugins?
- **RQ2.** How many plugins violate GDPR as reported by ChkPlug?
- **RQ3.** What is the computation overhead of ChkPlug?

The answer to the first question helps to show the effectiveness of ChkPlug, and the answer to the second question helps to understand the current compliance situation of plugins. Answering the third one will ensure the scalability of the tool.

B. RQ1: Performance Evaluation of ChkPlug

To evaluate the ability of ChkPlug to identify plugin violations, we compare the results of ChkPlug with manual analysis results. As mentioned in Section IV-B, we manually labeled 200 plugins as ground truth. In the following, we report the accuracy of ChkPlug by evaluating these 200 plugins.

We report the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) rates of the analysis in Table III. We consider violation as positive data and non-violation as negative data. TP indicates plugins labeled by both ChkPlug and the human annotator to be in violation, FP are labeled by ChkPlug to be in violation whereas manual analysis determines them to be compliant (non-violation), TN denotes plugins labeled by both ChkPlug and manual analysis to be compliant, and FN are labeled by ChkPlug to be compliant but human labels them as in violation.

We observe that our tool achieves an average of 98.8% TNR and 89.3% TPR in detecting four different GDPR violations. In the following, we discuss in detail about performance in each of those categories-

1) Evaluation of P_{access}: ChkPlug identified no plugins as violating an access policy. It is consistent with our manual analysis. ChkPlug achieves TNR of 100% and TPR of 100%. The main reason for all the plugins not violating access regulations is- whenever they store PII, they use WordPress core database. As a result, it is not strictly required to provide data export functionality to the user as WordPress will by default handle that.

2) Evaluation of P_{delete}: Out of the 200 plugins with ground truth from manual analysis, ChkPlug marks 66 plugins as violating a deletion policy with a TPR of 97% (127 out of 131) and a TNR of 96.2% (66 out of 69). We notice that a common source of false positives on the delete policy has to do with identifying deletion endpoints. While we can identify the static deletion endpoints made available to plugins

```
$request = wp_remote_get(
    'https://www.google.com/recaptcha/api/siteverify?secret='
    .$secret_key . '&response=' . $response .
    '&remoteip=' . $remote_ip
);
```

Listing 4: Security violations identified by CHKPLUG.

with a process similar to the one described in Appendix G, a developer is free to write their own implementation to adhere to the GDPR requirements. For these endpoints, CHKPLUG must fall back to pattern matching. In the false positive cases, the pattern matching is not exhaustive enough. For instance, in the plugin 'bp-featured-members', we observe PII being collected with a call to an add_user function. CHKPLUG correctly flagged this as a case where a deletion endpoint must be provided. However, the developer implemented deletion endpoint was defined as remove_user. This definition was easy to spot during a manual analysis, however, this form was not part of the patterns the tool searched for.

3) Evaluation of P_{share}: CHKPLUG marked 7 plugins as violating a sharing policy with a TNR of 99.5% (189 out of 190) and a TPR of 70% (7 out of 10). Our tool predicted one false positive. The 'wp-user-avatar' plugin, previously was sending user information to 'https://www.gravatar.com/avatar/' inside the 'class-wp-user-avatar-functions.php' file. However, while manually investigating this plugin, a human annotator noticed that the plugin had saved the deprecated code files inside a folder that is no longer used. Our tool is not able to detect which files are active and which are not. It analyzes all source files inside the plugin folder.

4) Evaluation of $P_{security}$: CHKPLUG marked 9 plugins as violating a security policy with a TNR of 99.5% (189 out of 190) and a TPR of 90% (9 out of 10). CHKPLUG analysis of security violations is conservative. When CHKPLUG cannot determine the endpoint receiving PII is using HTTPS, the corresponding plugin will be flagged as a violation. We see a case where this occurs in the 'google-site-kit' plugin. It tries to send PII using the url: $\$url = \$this->url(\$uri)$;. In the case of this plugin, the system was unable to resolve some of the components of the URL argument due to missing patterns. Even though manual analysis can identify the endpoint to be secure, since the full URL could not be resolved, it was marked as a violation. This is a case where CHKPLUG was over-conservative and led to a false positive.

C. RQ2: Measurement Analysis

We run CHKPLUG on 2,722 plugins from the WordPress store. Note that, these 2,722 plugins include the ones listed in Table II. We find that 14% (381 out of 2,722) plugins do not comply with GDPR.

Table V breaks down violations across particular policies. We observe that the most common form of violation was a delete violation, where users are not empowered to delete their data. Around 13.52% (368 out of 2,722) plugins do not provide data deletion functionalities. We observe the violations in other categories as 0.7% in P_{share}, 1.3% in $P_{security}$, and 0% in P_{access}. For the 19 plugins that violate P_{share}, we found that 16 of them do not have a privacy policy but share PIIs; the remaining three have a privacy policy but fail to mention PII sharing in that policy. Most of the plugins use the WordPress core database for storing data. Since WordPress provides the export feature for such storage, we experience fewer violations in P_{access}.

TABLE IV: Breakdown PIIs into violations based on the relations to the WP database. WP=wordpress core database, non-WP=custom database.

PII	#P_{delete}		#P_{share}		#$P_{security}$	
	WP	**non-WP**	**WP**	**non-WP**	**WP**	**non-WP**
Username	310	11	15	1	11	3
Email	29	3	2	0	5	4
Password	18	1	0	0	0	0
Address	12	1	2	0	0	0
First Name	12	0	2	0	3	3
Last Name	12	1	2	0	3	3
IP	7	2	1	0	5	4
State	8	0	2	0	1	0
Country	7	0	1	0	1	0
Phone	6	0	0	0	0	1
Postcode	4	0	1	0	1	0
City	4	0	1	0	1	0
Birthday	1	0	0	0	0	0

```
public function init() {
        if ( !class_exists( '\\GF_User_Registration' )
    !function_exists( '\\wlmapi_update_member' ) ) {
                // Gravity Forms User Registration Add-On
    and/or WishList Member not activated, so we do nothing
                return;
        }
    ....
}
....
private function insert_wlm_data( $user_id, $wlm_data ) {
        if ( empty( $wlm_data ) ) {
                return;
        }
        \wlmapi_update_member( $user_id, $wlm_data );
}
```

Listing 5: FP analysis of sending data to third-party.

TABLE V: RQ2: Measurement analysis results of 2,722 plugins using CHKPLUG.

Policy	#Violation	Percentage
P_{access}	0	0%
P_{delete}	368	13.52%
P_{share}	19	0.7%
$P_{security}$	36	1.3%

We also analyze the PIIs in GDPR violations. Examples of such PIIs are- phone number, date of birth, and physical address. We also break down PIIs in the GDPR violations based on data categories, and the relations to the WP database (called WP and non-WP). We show the details in Table IV.

We provide a few case studies of violations in Section VI.

D. RQ3: Computational Overhead

We benchmarked the runtime performance CHKPLUG on 2,722 plugins. The benchmarks ran on the same t3.medium machines described at the start of section V. No unnecessary user programs were running during the benchmark. We observe that the system can process an entire plugin in just a few minutes. The average time taken for analyzing each of the plugins is approximately 9.1 minutes. We run the benchmarks

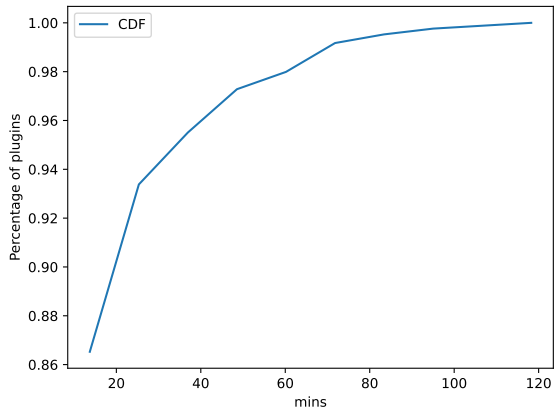


Fig. 4: RQ3: Cumulative distribution function (CDF) of Computation overhead of CHKPLUG.

with a three-hour timeout but no plugin has taken that long to be processed. The longest processing time is around two hours. In addition, we show in Figure 4 the cumulative distribution of plugins that take longer than the average time to analyze, i.e., the CDF of around 14% of plugins. We observe that more than 98% of plugins finish analysis within an hour.

We notice that the time it takes to process a plugin is proportional to its size. The larger the plugin is, the more processing time is needed. While we optimize the communication with the backing Neo4J database, batching reads and writes where possible, in profiling CHKPLUG, we discover that a large portion of the runtime is consumed by network overhead from these communications. This is a consequence of using an external database.

VI. CASE STUDY

In this section, we illustrate several case studies for the 381 plugins found by CHKPLUG, which violate GDPR policies. Specifically, we describe one violation example in each category. Note that we have responsibly reported all the violations to plugin developers; so far, we have not received any responses yet.

Violation of P_{delete} 'Namaste! LMS' (https://wordpress.org/plugins/namaste-lms/) provides a learning management system with an average rating of 4.5. Currently, it has 800+ active installations. It collects and stores user information in the database in file 'controllers/woocommerce.php' at line 62. It stores all PII in the WordPress core database. However, CHKPLUG does not find any evidence of providing users the option to delete those PII. It is in clear violation of P_{delete}.

Violation of P_{share} 'Vindi WooCommerce 2' (https://wordpress.org/plugins/vindi-payment-gateway/) provides a one-time payment solutions using Woocommerce subscriptions. It has been regularly maintained by Vindi (the developer of this plugin) with an active installation of 200+ users. At the time of writing this paper, it has an average rating of 3.7. Our tool detects this plugin as sharing data with a third party. In the 'controllers/CustomerController.php' file, it creates customer profile data. It also has the functionality to update customer data. Later, it sends that private data to the third party via API calls to https://sandbox-app.vindi.com.br/api/v1/ in file *src/services/Api.php* at line 242. Unfortunately, it does not even have a privacy policy. As a result, it violates GDPR by not disclosing PII sharing in the privacy policy.

Violation of $P_{security}$ 'Gallery Custom Links' (https://wordpress.org/plugins/gallery-custom-links/) allows linking images from galleries to a specified URL. It has 50,000+ active installations and an average rating of 4.5 stars. Our tool detects this plugin as violating $P_{security}$ because it is sending PII over an HTTP channel (to http://meowapps.com) instead of a secure channel. We find such problematic behavior in 'common/premium/updater.php' at line 418.

VII. DISCUSSION & LIMITATIONS

In this section, we summarize a few important discussions related to our data selection and tool evaluation, the limitations of our tool, adaptability to other CMS platforms, and future research directions.

WordPress Plugin Selection in the Evaluation While selecting the plugins for measurement analysis, we include the most popular ones as violations in these plugins would have the most impact on users. We use AWS for running all the evaluations and measurement analysis. It is expensive. Only around 1,000 plugins cost us $229.86 and it takes almost two and half days to finish running the analysis.

Deployment Model Website owners, plugin developers, and law professionals can use CHKPLUG to detect GDPR compliance. CHKPLUG will help website owners to identify plugins that violate GDPR laws. This will allow them to make informed decisions before integrating a particular plugin into their website. We have two mitigation suggestions for the plugin developers. First, developers can follow WordPress guidelines, which may help plugins to comply with many GDPR articles, e.g., data access, automatically. Second, our analysis report helps developers to make their plugins comply with GDPR. A sample report is shown in [15] Finally, law professionals can use our tool to help identify websites that violate GDPR laws. Note that, currently the WordPress team does not hold any responsibility for the GDPR compliance of the plugins. So, the website owner must take extra steps to ensure GDPR compliance with their website before integrating any WordPress plugins.

False Negatives Like all static analysis tools, CHKPLUG may suffer from lack of pattern coverage, leading to false negatives. However, from Section V-D, we can observe that our tool's false negative cases are rare. It shows that we are not missing many plugins which violate GDPR regulations.

False Positives Due to the inherent nature of static analysis, CHKPLUG has false positives, e.g., due to imprecise modeling of call edges. However, according to our evaluation results, CHKPLUG's TNR is very high.

CMS Platforms Other than WordPress We mainly perform our analysis on WordPress plugins as it is the most popular CMS powering over 32% of the World Wide Web [71]. However, our tool is extendable to plugins from other platforms (*e.g.,* joomla [13], drupal [10]) as long as those

are written using PHP, HTML, JavaScript. CHKPLUG will not be able to detect GDPR compliance in CMS platforms (*e.g.,* DotNetNuke [9], Kentico [14]) that use other programming languages (such as C# and ASP.net).

User Consents In our analysis, we have not considered user consent. As a result, we may miss plugins that collect and store PII without user permission. User consent involves users' interaction (via UI elements) and reading the terms & conditions document. It is context-dependent and there are many ways to collect user consent. It is another interesting research topic to analyze user consent in different contexts. We leave it for future work.

VIII. RELATED WORK

In this related work section, we start by describing previous works on GDPR analysis. Next, we present existing static analysis tools, particularly those on PHP and JavaScript, and how our tool is different from them. We end with discussing prior works on website privacy analysis.

A. GDPR Analysis

Investigating GDPR violence in published applications is a hot research topic. Researchers are performing experiments to detect whether the application includes proper privacy policy [27], whether they take user consent before storing user personal information [83]. To that end, they run several measurement analyses to have a deep understanding of the impact of GDPR across different regions [42] and how it evolves the existing application market [56]. A wide body of research works [48], [60], [75], [82] already discovered many problematic ways of granting cookie consent. Nouwens et al. [68] investigated 680 websites and found 90% of them do not comply with GDPR requirements. They pointed out several problems including – vague privacy policies [26], over-sharing of information through third parties [65]. In sum, the existing works are focusing more on making the usage of personal information more transparent [57], [63], [79], [81], [83], [84], which typically aim at generating GDPR compliant privacy policies from application's source codes [87], [89].

However, those works mainly focus on one platform while doing analysis. In contrast, CHKPLUG can detect GDPR violations across different programming languages. Specifically, we introduce an end-to-end detection tool which starts from how the application is collecting data from user, sending those data to the cloud, and how cloud process & stores those data.

B. Program Analysis

Program analysis, particularly either static or dynamic analysis, is a widely studied topic, and it likewise has been applied to a variety of situations, including but not limited to– code correctness, sanity checking, security, and privacy [28]. JFlow is an early application of static analysis to perform security and privacy for Java code [64]. Kim et al. [51] developed ScanDal, which statically analyzes Android apps for privacy leaks via tracking data flows from sinks to sources. AndroRisk is a risk quantifier for Android apps based on their requested device permissions along with dependent libraries [76]. Shezan et al. introduced TKPERM to transfer permission access knowledge across mobile, web and IoT platforms [78]. Xiao et al. applied static analysis to TouchDevelop scripts for Android, and these scripts were analyzed for private information sources flowing to sinks to outside sources as well as the security and privacy of such information flows [85]. Similar to those works, CHKPLUG can track personal information flow from the source to the sink. However, the purpose of CHKPLUG is to detect GDPR violations rather than direct privacy leaks or vulnerabilities.

Security researchers use query languages to identify vulnerabilities and other security bugs [43], [69]. They identified SQL injections and cross-site scripting in Java programs [58]. Martin et al. discovered functional flaws and security vulnerabilities using the Program Query language [59]. These days, researchers leverage graph-based program analysis to discover defects in the artifacts and identify zero-day vulnerabilities [25], [47]. Yamaguchi et al. leveraged code property graphs to discover vulnerabilities in the Linux kernel [86]. All of these works are focused on one programming language. Whereas, we create a cross-language code property graph to bridge the connection among different web programming languages.

Jalangi [77] selectively records and replays front- and back-end JavaScript programs. ODGen [55] design and proposes object dependence graph to detect Node.js vulnerabilities based on graph queries. DAPP [50] looks for AST and control-flow patterns for prototype pollution vulnerability detection. ObjLupAnsys [54] expands and maps two clusters during the abstract interpretation for vulnerability detection. Nodest [66], which is based on TAJS [45], detects command injection vulnerability via skipping unrelated packages. Black Widow [40] crawl and scan web applications in a black-box manner. People also proposed approaches to detect various client-side vulnerabilities, such as DOM-based XSS [61], [80] and CSRF [70] As a comparison, CHKPLUG's static analysis crosses language boundaries, e.g., PHP, JavaScript, SQL, and HTML.

C. Website Privacy Analysis

There exist several privacy enforcement tools, including – privacy preferences [36], privacy badger [29]. They are investigating various ways to improve user protection [52]. To that end, they enforce users' cookie policies on the visited websites by interacting with a consent management system [67], blocking first-party cookies (such as Google Analytics) [30]. Compared to those works, we are analyzing the source code of websites to find potential privacy leakages.

Most of the current work on WordPress plugin focus on measuring the vulnerabilities [33], [35], [62], [72], characterizing vulnerability exploits [34], [38], [53], [88]. Kasturi et al. investigate the impact of malicious plugins on CMS marketplaces [49]. Whereas, in our work, we investigate WordPress plugins to identify GDPR violations.

IX. CONCLUSION

WordPress plugins provide additional functionalities to websites built with WordPress but also face a new problem in the era of privacy, i.e., its compliance with privacy laws, particularly GDPR. To the best of our knowledge, no prior

works have provided automated checks of various GDPR articles on WordPress plugins partially due to its cross-language nature (the involvement of HTML, JavaScript, PHP, and SQL).

In this paper, we design a tool, called CHKPLUG, to automatically check whether WordPress plugins comply with GDPR. Ground truth evaluation shows that CHKPLUG performs well, achieving an average of 98.8% TNR and 89.3% TPR in checking GDPR compliance. We believe that website owners as well as plugin developers will get benefit by using CHKPLUG.

We hope that CHKPLUG can shed a light on the need for future research on checking CMS plugins' compliance against GDPR articles. More importantly, we believe that as the first step towards a more private, law-compliant community, CHKPLUG will help future researchers as well as developers to better understand GDPR and fill the gap between law enforcement and software development.

ACKNOWLEDGMENT

We would like to thank anonymous reviewers and shepherd for their helpful comments and feedback. This work was supported in part by National Science Foundation (NSF) grants 1920462, 1943100, 2114074, CNS-20-46361 and CNS-21-54404. Dr. Tian is supported by the Google research scholar award, Meta research award. Dr. Cao is also partially supported by Amazon Research Award 2021 and DARPA Young Faculty Award (YFA) 2022. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, Amazon, or DARPA.

REFERENCES

[1] Your introduction to personally identifiable information: What is PII? https://matomo.org/blog/2020/01/your-introduction-to-personally-identifiable-information-what-is-pii/, 2020.

[2] California consumer privacy act. https://oag.ca.gov/privacy/ccpa, September 16 2021.

[3] GDPR regulations. https://gdpr-info.eu/, 2021.

[4] Wordpress plugin. https://wordpress.org/plugins/, 2021.

[5] 2GDPR. https://2gdpr.com/, 2022.

[6] Calligo. https://www.calligo.io/gdpr-services-eu-representatives/, 2022.

[7] Codeinwp. https://www.codeinwp.com/blog/gdpr-compliance/, 2022.

[8] Cookiebot. https://www.cookiebot.com/, 2022.

[9] DotNetNuke. https://www.dnnsoftware.com/, 2022.

[10] Drupal. https://www.drupal.org/, 2022.

[11] HTML Parser. https://www.npmjs.com/package/htmlparser2, 2022.

[12] Is wordpress really a 10 billion dollar economy? https://www.presstitan.com/is-wordpress-really-a-10-billion-dollar-\economy/, 2022.

[13] Joomla. https://extensions.joomla.org/, 2022.

[14] Kentico. https://www.kentico.com/, 2022.

[15] Plugin analysis report generated by CHKPLUG. https://drive.google.com/file/d/119iZP0Ax6HuvX8YiTHXjqtAl1wkpXj1V/view, 2022.

[16] Secure privacy. https://secureprivacy.ai/, 2022.

[17] Silent breach. https://silentbreach.com/gdpr.php, 2022.

[18] SQL Parser. https://www.npmjs.com/package/node-sql-parser, 2022.

[19] Usercentrics. https://usercentrics.com/resources/gdpr-checklist/, 2022.

[20] What is: Theme. https://www.wpbeginner.com/glossary/theme/, 2022.

[21] Wordpress database functions. https://developer.wordpress.org/reference/classes/wpdb/, 2022.

[22] WordPress Plugin Handbook. https://developer.wordpress.org/plugins/privacy/, 2022.

[23] WordPress Plugin Handbook for Data Exporter. https://developer.wordpress.org/plugins/privacy/adding-the-personal-data-exporter-to-your-\plugin/, 2022.

[24] ALHUZALI, A., GJOMEMO, R., ESHETE, B., AND VENKATAKRISHNAN, V. {NAVEX}: Precise and scalable exploit generation for dynamic web applications. In *27th USENIX Security Symposium (USENIX Security 18)* (2018), pp. 377–392.

[25] ALRABAEE, S., SHIRANI, P., WANG, L., AND DEBBABI, M. Sigma: A semantic integrated graph matching approach for identifying reused functions in binary code. *Digital Investigation 12* (2015), S61–S71.

[26] ANDOW, B., MAHMUD, S. Y., WANG, W., WHITAKER, J., ENCK, W., REAVES, B., SINGH, K., AND XIE, T. {PolicyLint}: Investigating internal privacy policy contradictions on google play. In *28th USENIX security symposium (USENIX security 19)* (2019), pp. 585–602.

[27] ANDOW, B., MAHMUD, S. Y., WHITAKER, J., ENCK, W., REAVES, B., SINGH, K., AND EGELMAN, S. Actions speak louder than words:{Entity-Sensitive} privacy policy and data flow analysis with {PoliCheck}. In *29th USENIX Security Symposium (USENIX Security 20)* (2020), pp. 985–1002.

[28] BACKES, M., RIECK, K., SKORUPPA, M., STOCK, B., AND YAMAGUCHI, F. Efficient and flexible discovery of php application vulnerabilities. In *2017 IEEE european symposium on security and privacy (EuroS&P)* (2017), IEEE, pp. 334–349.

[29] BADGER, P. Electronic frontier foundation, 2019.

[30] BOLLINGER, D., KUBICEK, K., COTRINI, C., AND BASIN, D. Automating cookie consent and GDPR violation detection. In *31st USENIX Security Symposium (USENIX Security 22)* (2022), USENIX Association.

[31] BOSCH, J. From software product lines to software ecosystems. In *SPLC* (2009), vol. 9, pp. 111–119.

[32] BRIAN JACKSON. WordPress GDPR Compliance – Everything You Need to Know. https://kinsta.com/blog/wordpress-gdpr-compliance/#who-does-gdpr-impact, September 20, 2022.

[33] CABALLERO, J., GRIER, C., KREIBICH, C., AND PAXSON, V. Measuring {Pay-per-Install}: The commoditization of malware distribution. In *20th USENIX Security Symposium (USENIX Security 11)* (2011).

[34] CANALI, D., BALZAROTTI, D., AND FRANCILLON, A. The role of web hosting providers in detecting compromised websites. In *Proceedings of the 22nd international conference on World Wide Web* (2013), pp. 177–188.

[35] CERNICA, I., POPESCU, N., ET AL. Security evaluation of wordpress backup plugins. In *2019 22nd International Conference on Control Systems and Computer Science (CSCS)* (2019), IEEE, pp. 312–316.

[36] CRANOR, L. F. P3p: Making privacy policies more useful. *IEEE Security & Privacy 1*, 6 (2003), 50–55.

[37] DEGELING, M., UTZ, C., LENTZSCH, C., HOSSEINI, H., SCHAUB, F., AND HOLZ, T. We value your privacy... now take some cookies: Measuring the GDPR's impact on web privacy. *arXiv preprint arXiv:1808.05096* (2018).

[38] DUAN, R., ALRAWI, O., KASTURI, R. P., ELDER, R., SALTAFORMAGGIO, B., AND LEE, W. Towards measuring supply chain attacks on package managers for interpreted languages. *arXiv preprint arXiv:2002.01139* (2020).

[39] EDITORIAL STAFF. How Many WordPress Plugins Should You Install? What's too many? https://www.wpbeginner.com/opinion/how-many-wordpress-plugins-should-you-\install-on-your-site/, January 19, 2022.

[40] ERIKSSON, B., PELLEGRINO, G., AND SABELFELD, A. Black widow: Blackbox data-driven web scanning. In *2021 IEEE Symposium on Security and Privacy (SP)* (2021), pp. 1125–1142.

[41] FERRARA, P., AND SPOTO, F. Static analysis for GDPR compliance. In *ITASEC* (2018).

[42] GUAMÁN, D. S., DEL ALAMO, J. M., AND CAIZA, J. C. GDPR Compliance Assessment for Cross-Border Personal Data Transfers in Android Apps. *IEEE Access 9* (2021), 15961–15982.

[43] HALLEM, S., CHELF, B., XIE, Y., AND ENGLER, D. A system and language for building system-specific, static analyses. In *Proceedings of*

the ACM SIGPLAN 2002 Conference on Programming language Design and Implementation (2002), pp. 69–82.

[44] JASON COSPER. WordPress Themes: Overview and Tips on Finding the Perfect One. https://www.dreamhost.com/blog/how-to-find-wp-themes/, September 14, 2022.

[45] JENSEN, S. H., MØLLER, A., AND THIEMANN, P. Type analysis for JavaScript. In *Proc. 16th International Static Analysis Symposium (SAS)* (August 2009), vol. 5673 of *LNCS*, Springer-Verlag.

[46] JIA, Q., ZHOU, L., LI, H., YANG, R., DU, S., AND ZHU, H. Who leaks my privacy: Towards automatic and association detection with gdpr compliance. In *International Conference on Wireless Algorithms, Systems, and Applications* (2019), Springer, pp. 137–148.

[47] JOHNSON, A., WAYE, L., MOORE, S., AND CHONG, S. Exploring and enforcing security guarantees via program dependence graphs. *ACM SIGPLAN Notices 50*, 6 (2015), 291–302.

[48] KAMPANOS, G., AND SHAHANDASHTI, S. F. Accept all: The landscape of cookie banners in greece and the uk. In *IFIP International Conference on ICT Systems Security and Privacy Protection* (2021), Springer, pp. 213–227.

[49] KASTURI, R. P., FULLER, J., SUN, Y., CHABKLO, O., RODRIGUEZ, A., PARK, J., AND SALTAFORMAGGIO, B. Mistrust plugins you must: A large-scale study of malicious plugins in wordpress marketplaces. In *31th USENIX security symposium (USENIX security 22)* (2012).

[50] KIM, H. Y., KIM, J. H., OH, H. K., LEE, B. J., MUN, S. W., SHIN, J. H., AND KIM, K. DAPP: automatic detection and analysis of prototype pollution vulnerability in node. js modules. *International Journal of Information Security* (2021), 1–23.

[51] KIM, J., YOON, Y., YI, K., SHIN, J., AND CENTER, S. Scandal: Static analyzer for detecting privacy leaks in android applications. *MoST 12*, 110 (2012), 1.

[52] KONTAXIS, G., AND CHEW, M. Tracking protection in firefox for privacy and performance. *arXiv preprint arXiv:1506.04104* (2015).

[53] KOSKINEN, T., IHANTOLA, P., AND KARAVIRTA, V. Quality of wordpress plug-ins: an overview of security and user ratings. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing* (2012), IEEE, pp. 834–837.

[54] LI, S., KANG, M., HOU, J., AND CAO, Y. Detecting node.js prototype pollution vulnerabilities via object lookup analysis. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2021).

[55] LI, S., KANG, M., HOU, J., AND CAO, Y. Mining node.js vulnerabilities via object dependence graph and query. In *31st USENIX Security Symposium (USENIX Security 22)* (Boston, MA, Aug. 2022), USENIX Association.

[56] LINDEN, T., KHANDELWAL, R., HARKOUS, H., AND FAWAZ, K. The privacy policy landscape after the GDPR. *Proceedings on Privacy Enhancing Technologies 2020*, 1 (2020).

[57] LIU, F., WILSON, S., STORY, P., ZIMMECK, S., AND SADEH, N. Towards automatic classification of privacy policy text. *School of Computer Science Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-ISR-17-118R and CMULTI-17 10* (2018).

[58] LIVSHITS, V. B., AND LAM, M. S. Finding security vulnerabilities in java applications with static analysis. In *USENIX security symposium* (2005), vol. 14, pp. 18–18.

[59] MARTIN, M., LIVSHITS, B., AND LAM, M. S. Finding application errors and security flaws using pql: a program query language. *Acm Sigplan Notices 40*, 10 (2005), 365–383.

[60] MATTE, C., BIELOVA, N., AND SANTOS, C. Do cookie banners respect my choice?: Measuring legal compliance of banners from iab europe's transparency and consent framework. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 791–809.

[61] MELICHER, W., DAS, A., SHARIF, M., BAUER, L., AND JIA, L. Riding out DOMsday: Towards Detecting and Preventing DOM Cross-Site Scripting. In *Network and Distributed System Security Symposium (NDSS)* (2018). https://doi.org/10.14722/ndss.2018.23309.

[62] MESA, O., VIEIRA, R., VIANA, M., DURELLI, V. H., CIRILO, E., KALINOWSKI, M., AND LUCENA, C. Understanding vulnerabilities in plugin-based web systems: an exploratory study of wordpress. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1* (2018), pp. 149–159.

[63] MIAO, D. Y. *PrivacyInformer: An automated privacy description generator for the mit app inventor*. PhD thesis, Massachusetts Institute of Technology, 2014.

[64] MYERS, A. C. Jflow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (1999), pp. 228–241.

[65] NAN, Y., YANG, Z., WANG, X., ZHANG, Y., ZHU, D., AND YANG, M. Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in mobile apps. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2018).

[66] NIELSEN, B. B., HASSANSHAHI, B., AND GAUTHIER, F. Nodest: Feedback-driven static analysis of node.js applications. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2019), p. 455–465.

[67] NOUWENS, M., BAGGE, R., KRISTENSEN, J. B., AND KLOKMOSE, C. N. Consent-o-matic: Automatically answering consent pop-ups using adversarial interoperability. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts* (2022), pp. 1–7.

[68] NOUWENS, M., LICCARDI, I., VEALE, M., KARGER, D., AND KAGAL, L. Dark patterns after the GDPR: Scraping consent pop-ups and demonstrating their influence. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020), pp. 1–13.

[69] PAUL, S., AND PRAKASH, A. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering 20*, 6 (1994), 463–475.

[70] PELLEGRINO, G., JOHNS, M., KOCH, S., BACKES, M., AND ROSSOW, C. Deemon: Detecting csrf with dynamic analysis and property graphs. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2017), CCS '17, Association for Computing Machinery, p. 1757–1771.

[71] RANDY A. BROWN. WordPress vs Other CMS Platforms: How Does WordPress Stack Up Against the Rest? https://www.elegantthemes.com/blog/resources/wordpress-vs-other-cms-platforms-how-does-\wordpress-stack-up-against-the-rest, September 05, 2018.

[72] RUOHONEN, J. A demand-side viewpoint to software vulnerabilities in wordpress plugins. In *Proceedings of the Evaluation and Assessment on Software Engineering*. 2019, pp. 222–228.

[73] SAKAMOTO, T., AND MATSUNAGA, M. After gdpr, still tracking or not? understanding opt-out states for online behavioral advertising. In *2019 IEEE Security and Privacy Workshops (SPW)* (2019), IEEE, pp. 92–99.

[74] SANCHEZ-ROLA, I., DELL'AMICO, M., KOTZIAS, P., BALZAROTTI, D., BILGE, L., VERVIER, P.-A., AND SANTOS, I. Can i opt out yet? gdpr and the global illusion of cookie control. In *Proceedings of the 2019 ACM Asia conference on computer and communications security* (2019), pp. 340–351.

[75] SANTOS, C., BIELOVA, N., AND MATTE, C. Are cookie banners indeed compliant with the law? deciphering eu legal requirements on consent and technical means to verify compliance of cookie banners. *arXiv preprint arXiv:1912.07144* (2019).

[76] SARMA, B. P., LI, N., GATES, C., POTHARAJU, R., NITA-ROTARU, C., AND MOLLOY, I. Android permissions: a perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies* (2012), pp. 13–22.

[77] SEN, K., KALASAPUR, S., BRUTCH, T., AND GIBBS, S. Jalangi: A selective record-replay and dynamic analysis framework for javascript. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (New York, NY, USA, 2013), ESEC/FSE 2013, Association for Computing Machinery, p. 488–498.

[78] SHEZAN, F. H., CHENG, K., ZHANG, Z., CAO, Y., AND TIAN, Y. Tkperm: cross-platform permission knowledge transfer to detect over-privileged third-party applications. In *Network and Distributed Systems Security (NDSS) Symposium* (2020).

[79] SHEZAN, F. H., LAO, Y., PENG, M., WANG, X., SUN, M., AND LI, P. NL2GDPR: Automatically Develop GDPR Compliant Android Application Features from Natural Language. *arXiv preprint arXiv:2208.13361* (2022).

[80] STEFFENS, M., ROSSOW, C., JOHNS, M., AND STOCK, B. Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. In *Network and Distributed System Security Symposium (NDSS)* (2019). https://publications.cispa.saarland/id/eprint/2744.

[81] TESFAY, W. B., HOFMANN, P., NAKAMURA, T., KIYOMOTO, S., AND SERNA, J. I read but don't agree: Privacy policy benchmarking using machine learning and the eu GDPR. In *Proceedings of the The Web Conference* (2018), pp. 163–166.

[82] TREVISAN, M., TRAVERSO, S., BASSI, E., AND MELLIA, M. 4 years of eu cookie law: Results and lessons learned. *Proc. Priv. Enhancing Technol. 2019*, 2 (2019), 126–145.

[83] UTZ, C., DEGELING, M., FAHL, S., SCHAUB, F., AND HOLZ, T. (Un) informed Consent: Studying GDPR Consent Notices in the Field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 973–990.

[84] WADDELL, T. F., AURIEMMA, J. R., AND SUNDAR, S. S. Make it simple, or force users to read? paraphrased design improves comprehension of end user license agreements. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (2016), pp. 5252–5256.

[85] XIAO, X., TILLMANN, N., FAHNDRICH, M., DE HALLEUX, J., MOSKAL, M., AND XIE, T. User-aware privacy control via extended static-information-flow analysis. *Automated Software Engineering 22*, 3 (2015), 333–366.

[86] YAMAGUCHI, F., GOLDE, N., ARP, D., AND RIECK, K. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy* (2014), IEEE, pp. 590–604.

[87] YU, L., ZHANG, T., LUO, X., AND XUE, L. Autoppg: Towards automatic generation of privacy policy for android applications. In *Proceedings of the ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (2015), pp. 39–50.

[88] ZHOU, Y., WANG, Z., ZHOU, W., AND JIANG, X. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In *NDSS* (2012), vol. 25, pp. 50–52.

[89] ZIMMECK, S., GOLDSTEIN, R., AND BARAKA, D. Privacyflash pro: automating privacy policy generation for mobile apps. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2021).

[90] ZIMMECK, S., WANG, Z., ZOU, L., IYENGAR, R., LIU, B., SCHAUB, F., WILSON, S., SADEH, N., BELLOVIN, S., AND REIDENBERG, J. Automated analysis of privacy requirements for mobile apps. In *2016 AAAI Fall Symposium Series* (2016).

APPENDIX

We start with describing how WordPress plugin works. Next, we discuss detailed process of retrieving PII, identification technique of different functions, including- generic database operations, security functions used in security detector, remote request functions, PHP file opertation functions, WordPress database functions.

A. WordPress Plugin

Rapid development in web technologies has given web applications the opportunity to store, exchange and retrieve a significant information, some of which may be sensitive. This recent advance also made it possible to enhance software product line engineering methodologies such as plugin-based development [31]. The key idea is to offer developing additional features to extend the core capabilities. Generally, the core is maintained by some longterm developers while third-party developers build plugins around the core. To that end, the core provides some API and guideline to build plugins and themes. Later, people use those plugins in their website for the added functionalities. These plugins provide functionalities such as user registration, adding payment module, building social networks etc. We refer to user who uses those plugins as website owner (O).

TABLE VI: Retrieving personal data using Regex.

PII	Regex
email	.*(email).*
first name	.*(first.*name).*
last name	.*(last.*name).*
password	.*([^a-zA-Z]pass).
address	.*(address).
country	.*(country).*
state	.*([^a-zA-Z]state).
zipcode	.*(zipcode).
postcode	.*(postcode).*
city	.*([^a-zA-Z]city).*
birth	.*([^a-zA-Z]birth).*
username	.*(user.*name).
IP address	.*([^a-zA-Z]ip(.*address.*—[^a-zA-Z]—.*addr.*)).*
phone	.*(phone).*

Core offers several features (in the form of an API) that plugin developers can use to store and retrieve data. In order to accurately understand what data a WordPress function processes, stores, and alters, we model WordPress interfaces in this study. We scraped WordPress' official documentation for its functionalities in order to model its functions and methods. Both argument types and return types are included in this data. We manually labeled each function to indicate if it stores or retrieves personal information as well as what specific information is being stored (*e.g.,* user, post metadata, etc.). We added some additional set-based attributes to the function and method information so that we could perform set arithmetic to see if any data remained. Upon a trigger or manual activation, this system dynamically runs functions by name, and those action hooks return some data when manually invoked. Since this functionality is dynamic (actions are called by its registered name and thus is not static by nature), we modeled it as there are well-documented interfaces to add these hooks and triggers. This data is sometimes personal information and thus had to be modeled to obtain a comprehensive understanding of how a plugin handles personal information and whether it complies with GDPR regulations.

B. Retrieval of PII

We list the PII according to the existing article [1], [3]. We devise our rule-based approach for identifying PII in the user interface. Our intuition is that developer will ask PII in the form of an input field. We map the corresponding label with our manually crafted rules. We build this rule set based on investigating different website's interface (*e.g.,* registration page, login interface) which ask for user input. We have listed all the rules in Table VI.

C. Generic database operation

To analyze database operations, we need to carefully extract the SQL statements first. SQL statements are embedded inside the PHP code. By reading the PHP and WordPress documentation, we list PHP functions which are related to the SQL statements. Those functions are- dbDelta, exec, execute, get_col, get_results, get_row, prepare, query. We use these functions to search for database operations inside PHP code.

TABLE VII: Security Functions.

source	function	type	is state of the art?
php	crypt	encrypt	no
php	md5	hash	no
php	sha1	hash	no
php	password_hash	hash	yes
php	openssl_encrypt	encrypt	yes
php	openssl_digest	hash	yes
php	hash	hash	algorithm dependent
php	hash_hmac	hash	algorithm dependent
defuse	encrypt	encrypt	yes
phpseclib	encrypt	encrypt	yes
wordpress	wp_hash_password	hash	yes

TABLE VIII: Remote Request Functions.

function	language
$.post / jQuery.post	js
$.get / jQuery.get	js
$.ajax / jQuery.ajax	js
curl_exec	php
wp_remote_post	php
wp_remote_get	php
wp_remote_head	php
wp_remote_request	php

D. Security Functions

To identify $P_{security}$ violations, we need to investigate security functionalities that an individual plugins adapt. During pre-processing stage of building graph, we label all the nodes which involves encryption or hashing functionalities. Table VII demonstrates list the characteristics of all the security functions that we have considered in this study. According to GDPR, one should use state-of-the-art encryption technique to comply with $P_{security}$. That's why after listing all the encryption algorithm/functions, we mark those which are already broken by existing attacks. For example, at the time of writing this paper, md5 is not considered to be a safe encryption algorithm. So, during our traversal whenever we find nodes that perform encryption operations on PII, we mark that plugin as $P_{security}$ violation. Note that, the security of hash and hash_hmac depends on how they are implemented. Further analyzing the algorithm is out of scope of this work. For now, we only consider those as safe.

E. Remote Request

For analyzing $P_{security}$ and P_{share}, we need to identify whether the plugin is sending data to remote request or not. Because if it sends PII to remote request then it needs to follow security protocol to comply with $P_{security}$ and disclose such sharing in the privacy policy to avoid violating P_{share}. We model the graph in a way to handle and investigate the functions listed in Table VIII.

F. PHP file operation

One of the task of the sink detector is to detect whether the plugin store any PII in the file or not. On the other hand, our source detector is responsible for identifying the nodes which gets data from the database. To that end, we follow the PHP documentation to collect all the functions that can be used to set or retrieve data from a file. We list all of those functions and their types in Table IX. We label the function used for storing information as 'set'. Whereas, the functions which can be used to get data as 'retrieve'.

TABLE IX: Storing data in file using PHP functions.

function	action type
fwrite	set
file_put_contents	set
fputs	set
fputcsv	set
touch	set
fgetc	retrieve
fgetcsv	retrieve
fgets	retrieve
fgetss	retrieve
file_get_contents	retrieve
file	retrieve
fread	retrieve
fscanf	retrieve
readfile	retrieve

G. Wordpress database operation

We collect list of all the WordPress functions from the official website. In total, we list 2,885 functions. Out of this functions we manually label the type of functionalities of this functions. We find 30 storage functions, 65 retrieval functions and 28 deletion functions. We provide the full list in the following link- https://drive.google.com/file/d/1vy98kZGY91IIDlMrEjkBgMBzed1LYene/view?usp=sharing.